REMARKS

Claims 1-18 are pending in the present application. By this response, claim 19 is canceled; claims 1, 4, 9, and 17 are amended to clarify that the claimed invention is practiced in a data processing system and claim 18 is amended to include the features previously presented in claim 19. Reconsideration of the claims is respectfully requested in view of the above amendment and the remarks set forth below.

I. 35 U.S.C. § 101

The Office Action rejects claims 1-19 under 35 U.S.C. § 101 as being directed towards non-statutory subject matter. This rejection is respectfully traversed.

Claims 1-15 and 17 are rejected because they are allegedly not claimed to be practiced on a computer and therefore, the claims are not limited to practice in the technological arts. By this Response, independent claims 1, 4, 9 and 17 are amended to clarify that the claimed invention is practiced in a data processing system, and thus, are directed to statutory subject matter. Additionally, claims 16 and 18 recite a computer readable medium. Applicants respectfully submit that claims 16 and 18, in their present form are directed toward statutory material.

Furthermore, Applicants respectfully submit that each of the independent claims, un-amended, are directed toward statutory material. Each of the independent claims recites features that are inherently practiced on a computer such as a Web browser, a client and server side application, etc. Additionally, independent claims 1 and 4 recite entities such as "client side application portion" and "server side application portion". Further, independent claims 9 and 16 recite a "test client user interface" and along with independent claim 17, also recite a "workstation" which is itself a computer. Such entities offer further evidence that the presently claimed invention is practiced on a computer and therefore, is directed to statutory subject matter. Therefore, the claims as originally filed were directed to statutory subject matter.

Moreover, the Office Action fails to specifically address independent claims 16 and 18 with regard to the rejection under 35 U.S.C. § 101. Thus, Applicants respectfully

submit that independent claims 16 and 18 are allowable, not only because they are directed to statutory subject matter, but also because the Examiner has failed to address claims 16 and 18 in the rejection under 35 U.S.C. § 101.

In view of the above, Applicants respectfully request withdrawal of rejections to claims 1-19 under 35 U.S.C. § 101.

II. 35 U.S.C. § 112, First Paragraph

The Office Action objects to the specification under 35 U.S.C. § 112, first paragraph, as failing to adequately teach how to make and/or use the invention in claims 1-19. Additionally, the Office Action rejects the claims under the same reasons. This rejection is respectfully traversed. In rejecting these claims, the Office Action states:

> Claims 1-19 are rejected under 35 USC 112, first paragraph because current case law (and accordingly, the MPEP) require such a rejection if a 101 rejection is given because when Applicant has not in fact disclosed the practical application for the invention, as a matter of law there is no way Applicant could have disclosed *how* to practice the *undisclosed* practical application.

Office Action dated December 16, 2003, page 8.

Applicants direct the Examiner's attention to the remarks set forth above with regard to the rejection claims 1-19 under 35 U.S.C. § 101. Applicants respectfully submit that the rejection of claims 1-19 under 35 U.S.C. § 101 is overcome, and thus, the rejection of claims 1-19 under 35 U.S.C. § 112, first paragraph is also overcome.

III. 35 U.S.C. § 102, Alleged Anticipation of Claims 1-19

The Office Action rejects claims 1-19 under 35 U.S.C. § 102(e) as being allegedly anticipated by Helgeson et al. (U.S. Patent No. 6,643,652). This rejection is respectfully traversed.

With regard to claim 1, the Office Action states:

Claim 1's "providing a client side application portion for presenting a view to a user of a Web browser, said view allowing user interactions with said view, where some of said interactions specify given tests to perform on said component; and" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems. TM.JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the client-server system architecture with the development of JavaBeans. TM. and Java servlet generation. More recently, the technology...

Claim 1's "executing a server side application portion for receiving indications of said user interactions with said client side application portion and, responsive to said indications, performing said given tests on said component." is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

> Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems. TM.JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the client-server system architecture with the development of JavaBeans. TM. and Java servlet generation. More recently, the technology...

Claim 1, which is representative of claim 4 with regard to similarly recited subject matter, reads as follows:

> 1. A method of facilitating testing of an object-oriented application component, said method comprising:
> providing a client side application portion for presenting a view to a user of a Web browser, said view allowing user interaction with said view, where some of said interactions specify given tests to perform on said component; and
> executing a server side application portion for receiving indications of said user interactions with said client side application portion and, responsive to said indications, performing said given tests on said component. (emphasis added)

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). Applicants respectively submit that Helgeson does not identically show every element of the claimed invention arranged as they are in the claims. Specifically, Helgeson does not disclose specifying given tests to perform on a component and performing tests on the component.

Helgeson is directed to a system for managing data exchange among systems connected by a network. A system using a first type of framework is unable to transfer data to a system using a second type of framework, as systems implementing the first type of framework will have a different API than that of the second type of framework. In Helgeson, a data object is received from a first system in a first system specific type of format. The data object is then converted from the first type of format to a generic interchange format. The data object is then translated from the generic interchange format to a second system specific type of format. The data object is then transferred to the second system. (Summary)

Thus, Helgeson is concerned with data communications between multiple systems that have different API's. There is nothing in Helgeson that teaches specifying given tests to perform on a component and performing said given tests on the component. The Office Action alleges these features are taught at column 1, lines 52-67, which is reproduced above. This section discusses the use of object-oriented programming and more specifically the use of hardware independent platform languages to develop platform independent applications. Further, the section discusses how "the use of Java has been focused on the client side of the client-server system architecture with the development of JavaBeans ™ and Java servlet generation." Applicants fail to see any connection between this section of Helgeson and specifying given tests to perform on a component and performing said given tests on the component. This section of Helgeson does not even mention testing an object oriented application component, let alone providing a client side application portion for presenting a view to a user of a Web browser, wherein the view allows user interaction, where some of the interactions specify given tests to perform on the component and performing the tests on the components in response to an indication based on the user interactions.

While it may be true that this section of Helgeson teaches utilizing Java to develop platform independent applications and also mentions the phrase "client side" in the section reproduced above, there is simply nothing that even alludes to specifying given tests to perform on a component and performing said given tests on the component. This is not the problem Helgeson is concerned with. Helgeson is concerned with translating data between systems having different API's. Therefore, Helgeson is directed to an entirely different problem than the present invention. Thus, Helgeson does not teach or suggest specifying given tests to perform on a component and performing said given tests on the component.

With regard to claim 9, the Office Action states:

> Claim 9's "providing a test client user interface to a workstation over an HTTP link, where said test client user interface is viewed through the use of a Web browser run on said workstation;" is anticipated by Helgeson et al, col. 56, lines 19-30, where it recites:
>
> 1. Determines the SabaSite based on the request. The SabaSite is identified as follows:

a. Extract the servlet path information from the request object using the **HttpServletRequest API** (getservletpath()).

b. If the servlet path ends with a Web Content Server 800 specific string suffix, then the associated SabaSite name is determined by stripping of that suffix.

c. If the servlet path does not end with the Web Content Server 800 specific string suffix, then the system default SabaSite name is retrieved using the SabaSite API.

Claim 9's "receiving a selection from said workstation, said selection identifying a given object, where said given object is a home interface or a remote interface of said Enterprise JavaBean;" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems. TM.JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the client-server system architecture with the development of JavaBeans. TM. and Java servelet generation. More recently, the technology...

Claim 9's "receiving a request from said workstation, where said request is a consequence of user interaction with said test client user interface and includes an indication of a test to perform on said given object;" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers

have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems. TM.JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the client-server system architecture with the development of JavaBeans. TM. and Java servelet generation. More recently, the technology...

Claim 9's "responsive to said request, performing said test on said given object to give a result; and" is anticipated by Helgeson et al, col. 1, lines 52-67, where it recites:

Prior art systems of this type typically have an infrastructure which is tightly coupled to application products, specific hardware platforms and specific Operating systems and related services. Such systems are difficult to maintain, difficult to upgrade and difficult to extend to other applications as well as usually requiring redundant data input for their specific applications. In the past, developers have turned to object-oriented programming (OOP) to improve programming and code maintenance efficiency for such systems and to the use of hardware platform independent languages like Sun Microsystems. TM.JAVA.TM. language and system, as tools for developing such platform independent applications support systems. Until recently, the use of Java has been focused on the **client side** of the client-server system architecture with the development of JavaBeans. TM. and Java servelet generation. More recently, the technology...

Claim 9's "sending a response to said workstation over said HTTP link, said response including an indication of said result to be displayed by said user interface." is anticipated by Helgeson et al, col. 11, lines 39-67, where it recites:

Referring now to FIG. 4, the tier 3 applications server 307 is Expanded in FIG. 4 to illustrate the Business Applications Platform 415 of the present invention. In FIG. 4, the Platform contains an Interface Server 417, an Information Server 419, an Interconnect Server 423 and a Business Server 421. All of these Servers 417, 419, 421 and 423 may physically resign on the same hardware platform (such as a UNIX box or a Microsoft.TM. NT.TM. platform), or each server may reside on a separate hardware box, or any combination of servers and hardware boxes. Each of the servers may have included a JAVA Virtual Machine.TM. and the related

> runtime support. The electronic communications between these servers may use the XML protocol (409, 425, 427) with each server having services for translating XML into the particular Applications Programming Interface (API) language required by the server and for translating its internal language into XML prior to transmission to another server. In a preferred embodiment, all of these servers are contained in a single tier 3 platform, and may communicate with each other directly without the necessity of changing the interfacing protocol format. The Interface Server 417 (also alternatively designated herein as the WDK), communicates through a web server 405 via the internet 403 to web clients 401 via the HTML protocol. The Interface Server 417, also may communicate to a directly connected client 407 via other protocols such as XSL/XSLT etc., and may communicate to Personal Data Assistants 411 such as cell phones or Palm Pilots.TM. or other such wireless devices using wireless protocols such as WAP/WML, etc. The Interface Server 417,...

Claim 9, which is representative of claim 16 with regard to similarly recited subject matter, reads as follows:

> 9. At an application server, a method of facilitating testing of an Enterprise JavaBean, said method comprising:
>
> providing a test client user interface to a workstation over an HTTP link, where said test client user interface is viewed through the use of a Web browser run on said workstation;
>
> receiving a selection from said workstation, said selection identifying a given object, where said given object is a home interface or a remote interface of said Enterprise JavaBean;
>
> receiving a request from said workstation, where said request is a consequence of user interaction with said test client user interface and includes an indication of a test to perform on said given object;
>
> responsive to said request, performing said test on said given object to give a result; and
>
> sending a response to said workstation over said HTTP link, said response including an indication of said result to be displayed by said user interface. (emphasis added)

There is nothing in Helgeson that teaches performing tests on a component, as set forth above with regard to claims 1 and 4. For similar reasoning, Helgeson does not teach performing tests on a given object. There is absolutely nothing in the Helgeson that teaches testing of any type. The Office Action alleges this feature at column 1, lines 52-67, which is reproduced above. This section discusses the use of object-oriented

programming and more specifically the use of hardware independent platform languages to develop platform independent applications. While it may be true that Helgeson teaches utilizing Java to develop platform independent applications there is simply nothing that even alludes to an indication of a test to perform on a given object and performing a test on the given object, as recited in claim 9.

In addition, Helgeson also does not teach providing a test client user interface to a workstation over an HTTP link, where the test client user interface is viewed through the use of a Web browser run on the workstation. Nothing in Helgeson even alludes to a test client user interface. This is because Helgeson is not directed to anything having to do with testing. The Office Action alleges this feature is taught at column 56, lines 19-30 which is reproduced above. The HttpServletRequest API in Helgeson as highlighted in the Office Action is not comparable to the test client user interface of the present invention. First, an API is not a user interface. An API is an application program interface that is an interface between an application and an operating system. Secondly, there is nothing in Helgeson that teaches a client user interface specifically for testing. Thus, Helgeson does not teach all of the features recited in claims 9 and 16.

With regard to claim 17, the Office Action states:

> Claim 17's "select a given object;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:
>
> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "select a given method of said given object;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "supply said given method with a parameter;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "request that said given method be invoked with said parameter;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of

> the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "responsive to receiving said request, invoke said method with said parameter to give a result; and" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17's "present a further user interface to present said result to said user." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 17 reads as follows:

17. A Web module containing a test client for Enterprise JavaBeans, said test client operable to:

present a user interface over a data link, where said user interface may be displayed through the use of browser application on a remote workstation, said user interface allowing a user at said remote workstation to:

select a given object;
select a given method of said given object;
supply said given method with a parameter;
request that said given method be invoked with said parameter;
responsive to receiving said request, invoke said method with said parameter to give a result; and
present a further user interface to present said result to said user.(emphasis added)

Claim 17 recites the features of a user interface that allows a user to select a method from an object, supply the method with a parameter, invoke the method with the parameter and display the results of invoking the parameter on a user interface. This allows a user to supply test parameters to an object and obtain results from the test. Nowhere is anything of the kind taught in Helgeson. Helgeson has nothing to do with testing objects. As set forth above, Helgeson is concerned with translating data between systems having different API's. The Office Action alleges that Helgeson teaches the features recited in claim 17 at column 25, lines 28-43. However, the paragraph reproduced in the Office Action and above is actually located at column 32, lines 18-32. Thus, it is unclear which section is actually being referenced by the Office Action as allegedly teaching the features recited in claim 17.

In any case, neither section actually teaches the features recited in claim 17. The section reproduced above merely teaches some features of Java Naming and Directory Service (JNDI) including locating JavaBeans. JNDI is merely a part of the Java Enterprise API. JNDI has nothing to do with a user selecting a method from an object and supplying parameters to be invoked by the method. In addition, while column 25, lines 28-43 may discuss implementations of methods, nothing is taught regarding a user selecting a method, supplying a parameter to the method and executing the method with the user given parameter. Further, there is nothing that teaches a user interface to present the result from executing the method with the user given parameters to the user. There is simply nothing in the entire Helgeson reference that even alludes to the features recited in claim 17.

With regard to claim 18, the Office Action states:

Claim 18's "present a user interface over a data link, said user interface allowing a user to:" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "browse a Java Naming and Directory Interface namespace;" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "select a given object in said Java Naming and Directory Interface namespace; and" is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide

> **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18's "receive information regarding said given object." is anticipated by Helgeson et al, col. 25, lines 28-43, where it recites:

> As alluded to in the previous paragraph, before using a bean, it must first be located. All EJB application servers are required to provide **Java Naming and Directory Service (JNDI)** access for bean users. To use JNDI, a client application would typically first get an "initial context" (driven by properties such as where to find the **EJB server**, somewhat analogous to the JDBC connect string for locating a database), and then using the context, look up The home interface of the bean by its name. Using the home interface, the client can find a specific instance of a bean, create a new instance, or remove an instance. The naming service would be used and the interaction would be the same even if the bean instance is present locally (i.e., exists in the same Java Virtual Machine) instead of being deployed on a remote machine.

Claim 18, which is amended to incorporate the features of claim 19, reads as follows:

> 18. A computer readable medium containing computer-executable instructions which, when performed by a processor in an application server, cause the processor to:
>
> present a user interface over a data link, said user interface allowing a user to:
>
> browse a Java Naming and Directory Interface namespace;
>
> select a given object that is to be tested from said Java Naming and Directory Interface namespace;
>
> receive information regarding said given object; and
>
> perform a test on said given object. (emphasis added)

There is nothing in Helgeson that teaches testing a user-selected object as set forth above. The Office Action alleges that the features recited in claim 18 are taught at column 25, lines 28-43 of Helgeson. Applicants respectfully disagree and direct the Examiner's attention to the remarks set forth above with regard to this section of Helgeson. The section reproduced above merely discusses certain features of Java Naming and Directory Service (JNDI). JNDI has nothing to do with a user selecting a method from an object. Further, as noted above, the section reproduced above is actually located at column 32, lines 18-32, not column 25, lines 28-43 as stated in the Office Action. With regard to column 25, lines 28-43, while this section may discuss methods, nothing is taught regarding a user selecting an object to be tested. As noted above, there is simply nothing in Helgeson that teaches testing of an object-oriented application component.

In addition to the above, with regard to the rejection of claim 19, the Office Action alleges this feature is taught at column 36, lines 19-24, which reads as follows:

> In EJB specification 1.0, the deployment descriptor is a text file with a somewhat different format. The deployment descriptor is generally created using a GUI tool, generally supplied by EJB Server vendors. Additional information on deployment descriptors can be obtained from EJB literature and tool manuals.

This section merely teaches that a deployment descriptor is created with a graphical user interface tool supplied by Enterprise JavaBean Server vendors. This section has absolutely nothing to do with performing a test on a user-selected object. Thus, nowhere does Helgeson teach selecting a given object that is to be tested and performing a test on the given object.

In view of the above, Applicants respectfully submit that Helgeson does not teach each and every feature of independent claims 1, 4, 9 and 16-18 as required under 35 U.S.C. § 102(e). At least by virtue of their dependency on claims 1, 4, 9 and 16-18, Helgeson does not teach each and every feature of dependent claims 2-3, 5-8 and 10-15. Accordingly, Applicants respectfully request withdrawal of the rejection of claims 1-19 under 35 U.S.C. § 102(e).

In addition to the above, Helgeson does not teach or suggest all of the specific features recited in claims 3, 10 and 12. For example, with regard to claim 3, Helgeson does not teach that the runtime execution environment in which given tests on an Enterprise JavaBean are performed is the same runtime execution environment in which a server side application portion is executed. Helgeson does not teach performing tests on Enterprise JavaBeans. Applicants direct the Examiner to the detailed arguments set forth above regarding claims 1, 4, 9, 16 and 18. Specifically, nowhere does Helgeson teach performing a test on any type of object-oriented application, be it an Enterprise JavaBean, a component or an object. Moreover, Helgeson does not teach or suggest that tests are run on Enterprise JavaBeans in the same runtime execution environment in which a server side application portion is executed.

Further, the Office Action alleges this feature is taught at column 11, lines 39-67 of Helgeson. The Examiner apparently believes this section teaches the features of claim 3 merely because the section teaches that each of the servers have runtime support. There is no mention in this section, however, of anything having to do with performing tests on Enterprise JavaBeans, let alone doing so in the same runtime execution environment in which a server side application portion is executed. Thus, Helgeson does not teach all of the features of claim 3.

With regard to claim 10, Helgeson does not teach performing a test on a given object comprises invoking a method of the object. The Examiner alleges that this feature is taught at column 25, lines 28-43 of Helgeson. While this section may teach implementations of methods, there is nothing in this section or any other section of Helgeson that teaches invoking a method in response to a test being performed on a given object.

With regard to claim 12, Helgeson does not teach that a test client user interface provides a view that allows a user of a workstation to specify a particular JNDI server on which to allow the user to browse. The Office Action alleges this feature is taught at column 25, lines 28-43. Applicants direct the Examiner to the detailed discussion above regarding this section of Helgeson. In addition to the above, this section teaches locating an EJB server, not a JNDI server, as recited in claim 12.

Furthermore, Helgeson does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. Absent the Examiner pointing out some teaching or incentive to implement Helgeson to allow a user to perform testing on object-oriented application component, one of ordinary skill in the art would not be led to modify Helgeson to reach the present invention when the reference is examined as a whole. Absent some teaching, suggestion, or incentive to modify Helgeson in this manner, the presently claimed invention can be reached only through an improper use of hindsight using the Applicants' disclosure as a template to make the necessary changes to reach the claimed invention.

IV. Conclusion

It is respectfully urged that the subject application is patentable over Helgeson and is now in condition for allowance. The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

Respectfully submitted,

DATE: March 16, 2004

Stephen J. Walder, Jr.
Reg. No. 41,534
Carstens, Yee & Cahoon, LLP
P.O. Box 802334
Dallas, TX 75380
(972) 367-2001
Attorney for Applicants

SJW/kg